Task 4

Effects of Regularization Techniques

Alexander Garcia

May 2025

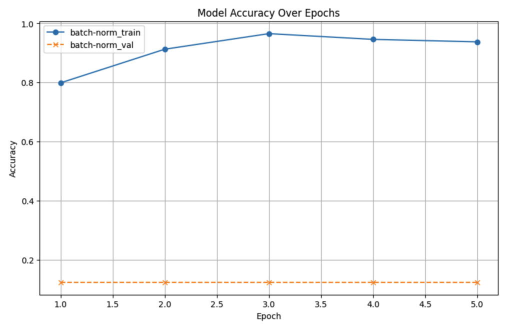
Contents

1 Introduction

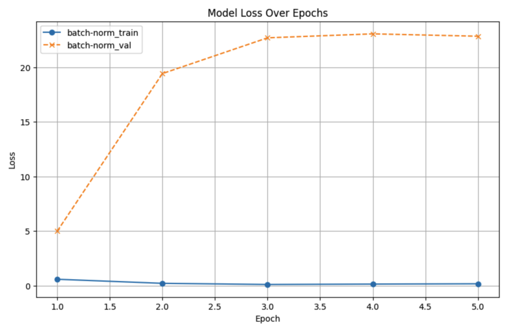
The goal of this phase is to test the effects of regularization techniques on the Clash of Clans data set. These include dropout, batch normalization, and regularizers. Normalization is a common technique in machine learning that tries to normalize the data over a range, usually done by subtracting the mean and dividing by the standard deviation. This centers the data around 0 and usually results in a fairly nice bell curve. It should be noted that Chollet says, “No one really knows for sure why batch normalization helps”. It is meant to normalize the data as it changes throughout training. Dropout randomly drops out input to reduce noise. Regularization adds penalty depending on which method is used but for this task I used L2 on the kernels.

2 Effects of BatchNormalization

The first attempt with BatchNormalization was after each Conv layer. However, this resulted in massive overfitting and the model unable to generalize to validation data as seen in the charts below



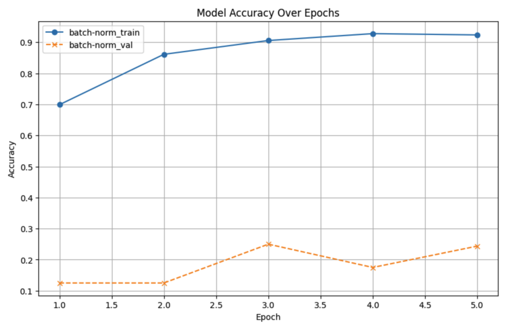
(a) Accuracy



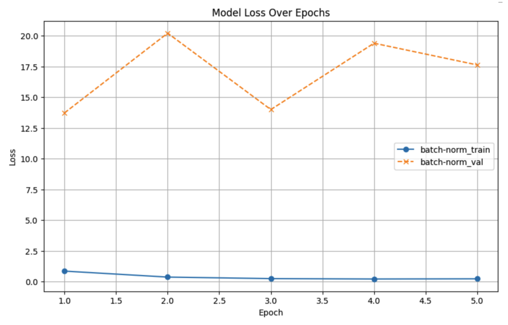
(b) Loss

Figure 1: Batch Normalization after every Conv

My second attempt was to reduce the amount of BatchNormalization layers and only have one at the end of the model just before entering the GlobalAveragePooling layer. However, this also resulted in overfitting.



(a) Accuracy

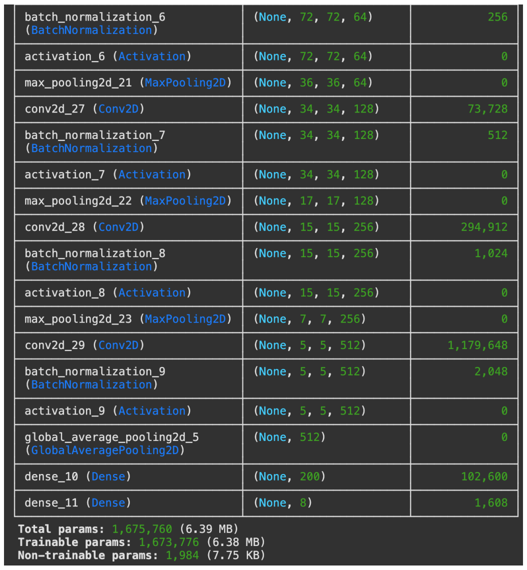


(b) Loss

Figure 2: 1 Batch Normalization layer

2.1 Tracking the Parameters of Batch Normalization

Unlike augmentation, batch normalization adds additional parameters. However, these can be set in Keras to be trainable parameters or non-trainable parameters. In any case, they do not add a large amount of parameters.



```
│ batch_normalization_6          │ (None, 72, 72, 64)     │           256 │
│ (BatchNormalization)           │                        │               │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ activation_6 (Activation)      │ (None, 72, 72, 64)     │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_21 (MaxPooling2D)│ (None, 36, 36, 64)     │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_27 (Conv2D)             │ (None, 34, 34, 128)    │        73,728 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_7          │ (None, 34, 34, 128)    │           512 │
│ (BatchNormalization)           │                        │               │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ activation_7 (Activation)      │ (None, 34, 34, 128)    │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_22 (MaxPooling2D)│ (None, 17, 17, 128)    │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_28 (Conv2D)             │ (None, 15, 15, 256)    │       294,912 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_8          │ (None, 15, 15, 256)    │         1,024 │
│ (BatchNormalization)           │                        │               │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ activation_8 (Activation)      │ (None, 15, 15, 256)    │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_23 (MaxPooling2D)│ (None, 7, 7, 256)      │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_29 (Conv2D)             │ (None, 5, 5, 512)      │     1,179,648 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_9          │ (None, 5, 5, 512)      │         2,048 │
│ (BatchNormalization)           │                        │               │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ activation_9 (Activation)      │ (None, 5, 5, 512)      │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling2d_5     │ (None, 512)            │             0 │
│ (GlobalAveragePooling2D)       │                        │               │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_10 (Dense)               │ (None, 200)            │       102,600 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_11 (Dense)               │ (None, 8)              │         1,608 │
└────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,675,760 (6.39 MB)
 Trainable params: 1,673,776 (6.38 MB)
 Non-trainable params: 1,984 (7.75 KB)
```

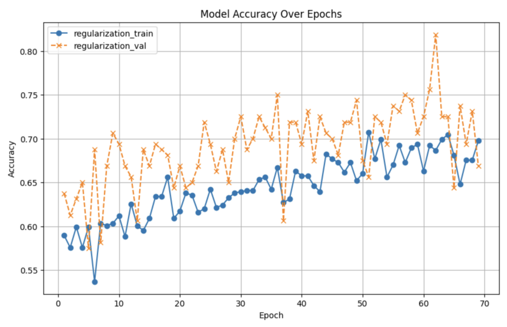
Figure 3: Parameters with normalization

3 Regularizers and Dropout

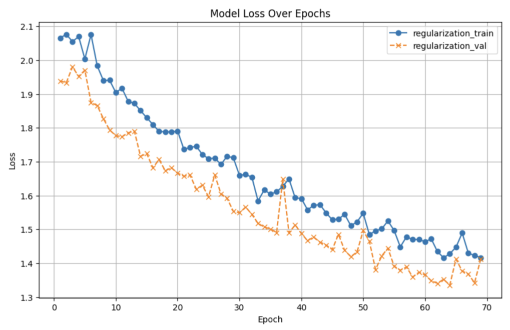
In this section I show the effects of adding kernel regularizers and a dropout layer in an attempt to increase validation accuracy. The first step was adding L2 regularizers to every layer and monitoring accuracy and loss as well as training time. Regularization adds penalty to the layer parameters. They can be applied to the kernel, bias, or activity (layer's output). For these examples I utilized kernel regularizers.

3.1 Effects of Regularizers and Dropout

Reported in these metrics is the first attempt with all conv layers being regularized. This drastically increased training time.



(a) Accuracy

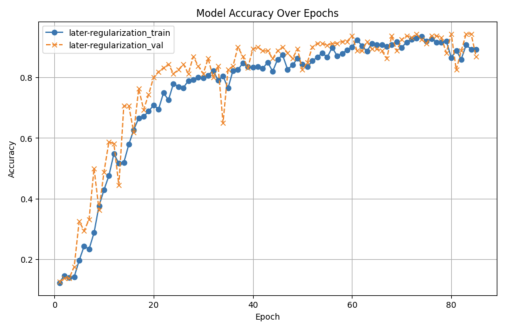


(b) Loss

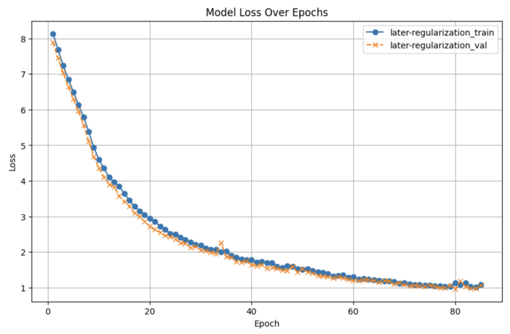
Figure 4: Regularized all layers

The model has previously been full of spikes in the curves but this is definitely a bit more drastic.

The second attempt involved only using regularization on the last layer which had 512 filters and the 200 Dense layer. The loss function has smoothed out while the accuracy still has learning spikes. Overall, the validation accuracy here did reach over 90% in the 80th epoch which was its best but reached early stopping at the 85th epoch. It did not perform better than the model without regularizers but exactly the same as both models reached their best validation accuracy at 94.38%. Overall the best effects have come from decreasing the learning rate on the adam optimizer and using "same" padding on the convolutions.



(a) Accuracy



(b) Loss

Figure 5: Regularized late layers

3.2 Table of Validation Accuracy

Regularization Type	**Accuracy / Result**
BatchNormalization all layers	Overfit
BatchNormalization last layer	Overfit
Regularized (L2, Dropout) all layers	75.63%
Regularized (L2, Dropout) last layers	94.38%
Normal model	94.38%

Table 1: Comparison of Regularization Strategies and Their Effect on Model Accuracy

The regularized (last layers) model and non-regularized model performed the best. However, when using regularization the model trains for nearly 80 minutes or longer even when using the T4 GPU on Google Colab.

4 Conclusion

In summary, I attempted to utilize various regularization methods to improve validation accuracy. I tested BatchNormalization to normalize data between the convolution layers but I was unable to benefit from this. The parameter count added seems inconsequential relative to the total amount of parameters this model has but the drastic overfitting was not a problem before adding batch normalization. The regularization did have a positive impact when only applied to the last convolution layer and first dense layer, but took significantly longer to train.